

# LAYSNET: A GRAPH-THEORETIC MARKOV MEMORY NETWORK FOR CPU-EFFICIENT CLASSIFICATION

Author: Ahmad Shajahan

Version: 0.2 research draft

## ABSTRACT

LaysNet is a graph-theoretic learning architecture that intentionally avoids the usual design of modern neural networks. It does not use dense layers, convolutional kernels, attention heads, biological neuron metaphors, or gradient descent as its core learning mechanism. Instead, it learns by building a directed graph memory. Inputs are transformed into numerical symbolic routes; nodes store label evidence; edges store Markov transition evidence; and prediction is performed by a semaphore router followed by smoothed probabilistic scoring. The reference implementation is pure Python, CPU friendly, and designed for transparent experimentation.

## 1. RESEARCH POSITION

The purpose of LaysNet is not to imitate the human brain. It is an attempt to define a different computational object: a probabilistic graph memory network. The term "network" here means a directed graph with stochastic transitions, not a biological neural network. The architecture treats each training example as a path through a discrete state space. The model learns the statistics of those paths and asks which label best explains a new route.

This paper frames LaysNet as an experimental research system. It is new as an implementation and architecture in this repository, but the correct scientific position is careful: it must be benchmarked broadly before any state-of-the-art claim is made.

## 2. MOTIVATION

Backpropagation-based neural networks are powerful, but they can be opaque and expensive. Small CPU machines often cannot train image models comfortably. LaysNet explores a different design target: make the learned object inspectable, keep training simple, and allow a prediction to be explained as a route through graph memory.

The model is inspired by graph theory, Markov chains, stochastic processes, and numerical encoding. It replaces weights in hidden layers with counts in graph nodes and directed edges. Every prediction can expose its encoded symbols, route, class log-scores, and Euler balance diagnostic.

### 3. ARCHITECTURE SUMMARY

The pipeline is:

input vector

- > feature standardization
- > deterministic projection encoder
- > quantile numerical symbols
- > semaphore graph router
- > node memory likelihoods
- > Markov edge likelihoods
- > class prediction

During training, the model stores node-label counts and edge-label counts. During prediction, it maps the input to a graph route and scores each class with smoothed log probabilities.

#### 4. INPUT SPACE

Let a sample be:

$x$  in  $\mathbb{R}^d$

Each feature is standardized using training statistics:

$$x'_j = (x_j - \mu_j) / \sigma_j$$

This is not a learned neural layer. It is a deterministic preprocessing step that prevents large-scale dimensions from dominating the graph route. For images, the input can be a flattened grayscale vector. For tabular data, it is the numeric feature vector.

## 5. PROJECTION ENCODER

For route length 'T', LaysNet constructs deterministic projection vectors:

$$r_t \text{ in } \mathbb{R}^d, t = 1..T$$

The raw code at stage 't' is:

$$z_t = \langle x, r_t \rangle$$

The reference implementation uses structured trigonometric projections with seeded jitter. This creates repeatable but different routes for different seeds. Future work can replace this encoder with learned or domain-specific graph encoders.

## 6. QUANTILE NUMERICAL ENCODING

Each raw projection value is converted into a discrete code:

$$c_t = Q_t(z_t) \text{ in } \{0, 1, \dots, B - 1\}$$

The function 'Q\_t' is learned from training quantiles at stage 't'. The encoded input is:

$$C(x) = (c_1, c_2, \dots, c_T)$$

This is the main numerical encoding. It transforms continuous vectors into symbols that can live as graph nodes.

## 7. GRAPH NODES

Every encoded stage becomes a graph node:

$$v_t = (t, c_t)$$

The stage index matters. Code '5' at stage '2' is not the same node as code '5' at stage '9'. A training example becomes a path:

$$R(x) = (v_1, v_2, \dots, v_T)$$

The node memory records how often each class visits each node:

$$N(v, y)$$



## 8. GRAPH EDGES

Adjacent nodes define directed edges:

$$e_t = (v_t \rightarrow v_{t+1})$$

The edge memory records how often each class traverses each transition:

$$M((u, v), y)$$

Edges are the Markov part of the network. They store directional route evidence, not just isolated feature buckets.

## 9. SEMAPHORE ROUTER

When a new input produces a node that was seen during training, prediction uses that exact node. If the code was not seen at that stage, the semaphore router sends the route to the nearest known code:

$\text{route}(t, c) = (t, c)$  if  $c$  is known at stage  $t$   
 $\text{route}(t, c) = (t, \text{argmin}_k |k - c|)$  otherwise

The term semaphore is used because the router controls which graph path is allowed for an incoming encoded signal. It is deterministic and inspectable.

## 10. NODE PROBABILITY

For label set 'Y' and additive smoothing 'alpha', the node-label probability is:

$$P_y(v) = \frac{(N(v, y) + \alpha)}{(N(v) + \alpha |Y|)}$$

This measures whether the current route node historically supports a class. Smoothing prevents zero probability when evidence is sparse.

## 11. MARKOV EDGE PROBABILITY

The class-conditioned transition probability is:

$$P_y(v_{t+1} | v_t) = \frac{(M((v_t, v_{t+1}), y) + \alpha)}{(M((v_t, v_{t+1}))) + \alpha |Y|}$$

This is a finite Markov chain over graph states. The transition is not globally learned as a dense matrix; it is stored only where training routes create evidence.

## 12. PREDICTION OBJECTIVE

For route 'R', LaysNet scores each label:

$$\begin{aligned} S(y \mid R) = & \\ & \log P(y) \\ & + \lambda_{\text{node}} * \sum_t \log P_y(v_t) \\ & + \lambda_{\text{edge}} * \sum_t \log P_y(v_{t+1} \mid v_t) \end{aligned}$$

The prediction is:

$$y_{\text{hat}} = \operatorname{argmax}_y S(y \mid R)$$

This is the central mathematical engine of the current implementation.

### 13. EULERIAN DIAGNOSTIC

For route 'R', define directed in-degree and out-degree inside the route. The Euler balance error is:

$$E(R) = \sum_v |\text{outdeg}_R(v) - \text{indeg}_R(v)|$$

A perfect Eulerian circuit has zero imbalance. LaysNet does not require such a route, but the diagnostic can be used to study route stability, graph compression, and future Euler-inspired regularization.

## 14. STOCHASTIC PROCESS VIEW

LaysNet can be read as a stochastic process over a finite graph. The current state is the routed node. The next state is selected by the encoded input and evaluated through class-conditioned transition evidence. The probability model is not generative in the full image sense; it is a discriminative route scorer built from stochastic graph counts.

This gives a different research direction from deep neural representation learning. It asks whether useful classification can come from symbolic graph routes and smoothed transition statistics.

## 15. ENSEMBLE VARIANT

The optional ensemble trains multiple LaysNet graph memories with different projection seeds. If member 'm' produces score 'S\_m(y)', the ensemble computes:

$$S_{\text{ensemble}}(y) = (1 / M) \sum_m S_m(y)$$

The ensemble remains graph based. It combines Markov route evidence and does not introduce a dense hidden layer. In the current Iris split, the single tuned LaysNet member is strongest; the ensemble is useful as a stability experiment.



## 16. COMPUTATIONAL COMPLEXITY

Let 'n' be training size, 'd' input dimension, 'T' route stages, and '|Y|' class count.

training projection:  $O(nd)$

training memory:  $O(nT)$

prediction:  $O(Td + T|Y|)$

memory:  $O(\text{unique nodes} + \text{unique edges})$

The implementation uses counters and sparse graph storage. This is why it can run on small CPU machines.

## 17. EXPERIMENTAL PROTOCOL

The repository includes a real small benchmark using the UCI Iris dataset. The script compares LaysNet with majority class, k-nearest neighbors, and Gaussian naive Bayes on the same stratified train/test split.

The default local command is:

```
python3 examples/compare_iris.py
```

The generated report is saved to:

```
reports/iris_comparison.md
```

No fabricated scores should be added to the paper.

## 18. CURRENT MEASURED RESULT

On the default local split used by the repository, the measured comparison is:

GaussianNB	1.0000
LaysNet	1.0000
LaysNetEnsemble	0.9778
kNN-k3	0.9778
Majority	0.3333

This is a small benchmark, not proof of universal superiority. The scientific result is that LaysNet can be competitive on a classic small dataset while remaining graph based and inspectable.

## 19. IMAGE PIPELINE

For image classification, the reference script expects a folder structure:

```
data/animals/train/cat  
data/animals/train/dog  
data/animals/test/cat  
data/animals/test/dog
```

Images are converted to grayscale, resized, flattened, normalized to '[0, 1]', and passed to LaysNet. This gives a simple cat/dog experiment path for Kaggle datasets without requiring a GPU.

## 20. LIMITATIONS AND SAFETY OF CLAIMS

LaysNet is not a replacement for all deep learning. Quantile encodings can lose detail. Large images may need better encoders. Graph memory can grow with route diversity. The model has not been evaluated across enough datasets to claim state-of-the-art status.

The right research language is: LaysNet is a new graph-probabilistic architecture prototype with working code, tests, documentation, and reproducible small comparisons.

## 21. FUTURE WORK AND CONCLUSION

Future work should explore learned encoders, graph pruning, thermodynamic-style annealing for route compression, C extensions for projection speed, sparse serialization, and larger public benchmarks. The core idea is to make classification a problem of graph route memory and stochastic transition scoring.

LaysNet is therefore a different kind of neural-network-like system: not biological, not backpropagation centered, and not dense-layer based. Its contribution is a transparent path from numerical encoding to graph memory to Markov prediction.

## REFERENCES

- Fisher, R. A. The use of multiple measurements in taxonomic problems, 1936.
- UCI Machine Learning Repository, Iris dataset.
- Python Packaging User Guide.
- PyPI Trusted Publishers documentation.